AMENDMENTS TO THE SPECIFICATION

In the Specification:

Please replace the paragraph beginning on page 2 line 3 starting with "Conventionally, there are" with the following amended paragraph:

Conventionally, there are two divergent approaches to persistence in software test management. Test management data can be stored in a relational database, optimized for querying and reporting. For example, Mercury TestDirector® and IBM Rational TestManager® utilize this approach. However, this creates a problem in that the database reflects a snapshot in time and as a consequence tests and source cannot be kept in sync unless all development assets are backed up (baselined) at once, which is typically only done sparingly (e.g., product ships, beta releases…). This has been the favored approach for managing testing activities in teams. An alternative approach is to store tests as source code with granular version control consistent with the source code. For instance, open source frameworks NUnit® (for .Net™) and JUnit® (for Java®) utilized this approach. This allows tests and source under test to be synchronized but prevents the querying and reporting that are necessary for managing a testing effort across a team of any size. Accordingly, this approach has been used for testing performed by individual programmers on their own code, but not employed for team activity.

Please replace the paragraph beginning on page 5 line 22 starting with "Turning initially to Fig. 1" with the following amended paragraph:

Turning initially to Fig. 1, a test management system 100 is illustrated in accordance with an aspect of the present invention. Test management system 100 comprises software under test (SUT) component 110, test case component 120, version component 130, and test case file component 140. The test management system 100, at least in part, provides a system or subsystem that facilitates coherent interaction between and amongst developers and testers over a software lifecycle. Accordingly, it should be appreciated that although not shown developers and testers may interact with various test management system components to view data and/or

effectuate changes. SUT component 110 includes and refers to a software application that has been developed or is in the process of being developed by one or more developers. A developer can be a person that authors product source code. Furthermore, it is to be appreciated that SUT can also refer to System Under Test as the software in fact defines a unique computing machine or system. Test case component 120 corresponds to a formal description of an individual test perhaps including but not limited to test script(s), test input/stimulus, and test conditions needed for execution of the test. The test management system 100 also includes a version component 130. Version component 130 monitors both the SUT component 110 and the test case component 120 for changes. For example, component 130 can detect a version change in the source code under test. This can, *inter alia*, help ensure the elimination of many false positive results that may otherwise occur during testing if it was not so noted that the source under test had changed, for instance.

Please replace the paragraph beginning on page 6 line 26 starting with "Turning to Fig. 2a" with the following amended paragraph:

Turning to Fig. 2a, a test catalog 210 is illustrated in accordance with an aspect of the subject invention. The test catalog 210 provides a repository for a collection of test case files 140, test cases 120, test variations 220, and test suites 230, inter alia. The test catalog ~~design~~ 210 provides a data store layout which is simple, extensible, and scalable as well as enabling, as users can manage a central store and/or their own local store using familiar concepts. The test catalog ~~store~~ 210 can be based on simple file storage, extended to create a hierarchical store. In particular, the test catalog 210 can be constructed from the aggregation of individual files (e.g., TCX files), which relate to each other in a hierarchical fashion. A single file, such as a TCX file (also referred to herein as test case file), is the standard unit for a simple test catalog. The TCX file is a complete store for namespace metadata, test case, and test suite metadata, as well as their namespace relations.

Please replace the paragraph beginning on page 7 line 20 starting with "Returning to Fig. 2a" with the following amended paragraph:

Returning to Fig. 2a, it should be appreciated that test catalog 210 can be specified in the form of a high performance database (e.g., SQL server™) or as a text file (e.g., XML catalog for revision control). According to an aspect of the subject invention a user can create a test catalog, save changes to a test catalog, as well as import or export the test catalog or a subset thereof (e.g., to an XML file- TCX file).

Please replace the paragraph beginning on page 7 line 25 starting with "As shown in Fig. 2a" with the following amended paragraph:

As shown in Fig. 2a, exemplary test catalog 210 comprises a plurality of test case components 120 (Test Case Component1, Test Case Component 2 through Test Case ComponentN, where N is an integer greater than or equal to one). Test cases 120 can be formal definitions of individual tests including but not limited to test script(s), test input/stimulus, and test conditions necessary for the execution of the described test. A test author can create and customize tests cases 120 in a multitude of different manners. For example, a tester can create a test case entry and specify its hierarchy in the test catalog 210. In addition, the test case type can be specified (e.g., manual, automatic, unit, load, functional…). It should also be appreciated that once the test case is specified it can later be edited or copied. For instance, a hierarchy sub-tree can be copied to another location in the hierarchy, test cases can be associated with requirements, test binaries, or a test project. Test variations 220 can also be included in the test catalog. Test variations 220 refer to a test artifact representing a test case with partially specialized parameter binding. Test variations 220 can be generated manually or automatically, for instance with a test rig. A test rig as used herein refers to hardware and/or software utilized to host a text execution. Finally, test catalog component 210 can also comprise one or more test suite components 230. Test suite[[s]] component 230 can be collection of test components (e.g., multiple test cases, hierarchy of test cases…).

Please replace the paragraph beginning on page 8 line 12 starting with "To facilitate a clear" with the following amended paragraph:

To facilitate a clear understanding of the interaction between particular test components Fig. 3 has been provided. Fig. 3 is a schematic block diagram 300 illustrating the key data component relationships in accordance with an aspect of the subject invention. ~~Source under test~~ [[()]SUT[()]] component 110 represents the computer system source code. The SUT component 110 implements such functionality on a computer system, which provides particular ~~features~~ feature(s) 320. Furthermore, such feature(s) 320 can be implemented by way of a work item 330. In other words work items 330 are implemented by features of the SUT component 110. Test case component 120 provides tests including but not limited to scripts and input/simulation data to examine or test the SUT component 110. The SUT component 110, the feature(s) 320 and the test case component 120 all represent versioned data. Stated somewhat differently, each component can and often does change. During the development process code represented by SUT component 110 is continuously modified such that new features are added and/or removed. Furthermore, test case components 120 need to change to test the changes in the source under test 110. Test case component 120 generates test results 350. Test results 350 cover or correspond to the results of the test as executed on the current version of SUT component 110. Version component 130 monitors and records changes to the SUT component 110. Build drop component 370 comprises the executable version of the software under test. The build drop component 370 includes a changed data from the version component. Accordingly, the test results, the version component 130 changes, and the build drop are all version tagged data, meaning that they are all dependent on the version of the software under test.

Please replace the paragraph beginning on page 90 line 3 starting with "Fig. 4 depicts"with the following amended paragraph:

Fig. 4 depicts a storage system 400 in accordance with an aspect of the subject invention. Workspace 410 defines a boundary for isolation between transacted changes and version control. Application 412 provides instructions for executing tasks 414 on work items or data 416. Application 412 comprises a plurality of sections and tests integrated therein to facilitate synchronization. Version component 130 provides for source code version control. In essence, version component 130 monitors and tracks changes to the application 412 including tests residing therein. Test catalog 210 includes, inter alia, test cases, each test case having properties

and file associations (e.g., paths). The test catalog 210 can be based on simple file storage, extended to create a hierarchical store. In particular, the test catalog 210 can be constructed from the aggregation of individual files such as XML files (also referred to herein as TCX (Test Case XML) files). A development team can share a centralized test catalog 210, which can be a database executing SQL server, for instance. Alternatively, individual members can have their own local test catalog 210, if desired. The test catalog 420 can be loaded with tests and other data from application 412. It should be appreciated that persistence in a TCX file in the test catalog 210 allows versioning consistent with a source as well as version-aware reference to a source under test. Application 412 can also be published to drop folder(s) 422 to provide a single reference point for testing. Additionally, test catalog 210 [[416]] can interact with drop folder(s) 422 to archive and reload source code and test cases thereby supporting reversion. Text information from test catalog 210 [[420]] and application source code residing in drop folder(s) 422 can be provided to a test execution component 424, which executes specified tests on the application source code. Subsequently, the test execution component 424 can provide test results 350. The test results 350 are then tagged and saved to XML files 428 which can thereafter be published to the enterprise data store 430. Furthermore, test catalog data 210 and test results 350 can be published directly to the enterprise data store 430 as version tagged data for historical and trend analysis. Additionally, the enterprise data store 420 can store drop folder data, source code, and work items.

Please replace the paragraph beginning on page 10 line 17 starting with "To facilitate a clear" with the following amended paragraph:

Fig. 5 depicts a method of test management 500 in accordance with an aspect of the subject invention. At 510, metadata is retrieved regarding test version information relating to a source or software under test. In other words, what is received is data defining current relationships between tests and source code under test such that it can be determined whether a test tests [[a]] the current version of code or an old version of the code, and how the test results relate to the present code under test. Thus, if a test relates to a particular code version then results generated by the test on subsequent source code versions may not be comparable because of the changes thereto. Furthermore, it should be appreciated that at least a portion of the

metadata regarding versions can come from a version component or conventional source code control system. Still further yet it should be appreciated that according to an aspect of the present invention such version metadata can be employed in daily builds and testing. At 520, the metadata is persisted to a mark up language file such as XML, which provides a mechanism for defining or marking up data. At 530, test result data is version tagged to enable expeditious review of results with respect to the source code tested and the tests thereon. Storage of information in the manner described provides for explicitly specifying test source code relationships as well as easy data queries, for example via XSLT transformations.